S-learning: A model-free, case-based algorithm for robot learning and control

Brandon Rohrer

Sandia National Laboratories, Albuquerque, NM 87185, USA,
brrohre@sandia.gov

Abstract. A model-free, case-based learning and control algorithm called S-learning is described as implemented in a simulation of a light-seeking mobile robot. S-learning demonstrated learning of robotic and environmental structure sufficient to allow it to achieve its goal (reaching a light source). No modeling information about the task or calibration information about the robot's actuators and sensors were used in S-learning's planning. The ability of S-learning to make movement plans was completely dependent on experience it gained as it explored. Initially it had no experience and was forced to wander randomly. With increasing exposure to the task, S-learning achieved its goal with more nearly optimal paths. The fact that this approach is model-free and case-based implies that it may be applied to many other systems, perhaps even to systems of much greater complexity.

1 Introduction

S-learning is a general learning and control algorithm modeled on the human neuro-motor system [2, 8, 9]. It is model-free in the sense that it makes no assumptions about the structure or nature of the system being controlled or its environment. S-learning accomplishes this using case-based reasoning. It uses previous experience to help it select actions. This paper describes the implementation of S-Learning in computer code and the application of S-learning to a simulated mobile robot.

1.1 Relation to Previous Work

Most approaches to robot control assume the existence of an explicit system model. Even the majority of learning algorithms take the form of a search in parameter space, with the underlying structure determined beforehand. Other methods make a less constraining assumption: that the vectors of state information occupy a metric space. This allows the use of distance metrics, such as the L^2 norm, to interpret its state history. These include finite state machines [11], variants of differential dynamic programming [7, 12], the Parti-game algorithm [6], and probabilistic roadmaps [3]. But even this seemingly benign assumption implies a good deal about the system being modeled. It is violated by

any sufficiently non-smooth system, such as one containing hard-nonlinearities or producing categorical state information.

There are still a number of algorithms that are similar to S-learning in that they make no assumptions about the system being learned and controlled. These include Q-learning [13], the Dyna architecture [10], Associative Memory [4], and neural-network-based techniques including Brain-Based Devices [5] and CMAC [1]. These approaches, together with S-learning, can be categorized as reinforcement learning (RL) algorithms, or solutions to RL problems. However, these all assume a static reward function, where S-learning does not.

1.2 Dynamic Reinforcement Learning Problem Statement

To be more precise, S-learning addresses a general class of reinforcement learning (RL) problem, referred to hereafter as the dynamic RL problem: how to maximize reward in an unmodeled environment with time-varying goals. More specifically, given discrete-valued action (input) and state (output) vectors, $a \in \mathcal{A}$ and $s \in \mathcal{S}$, and an unknown discrete-time function f, such that

$$s_t = f(a_{i \leq t}, s_{i<t}, t), \tag{1}$$

(where the notation $a_{i \leq t}$ denotes the set of all a_i such that $i \leq t$) and a scalar reward, r, and known reward function, g, such that

$$r_t = g(s_{i \leq t}, t), \tag{2}$$

maximize the total reward over time:

$$V = \sum_{i=0}^{\infty} r_i \tag{3}$$

Equation 3 shows an infinite-horizon formulation, but finite- and receding-horizon variations of the dynamic RL problem are similarly structured.

The dynamic RL formulation is relevant to a large class of problems. It is applicable in instances where 1) the model is unavailable and 2) the reward function varies with time. Models may be unavailable for a number of reasons. Systems may be too complex to model accurately with the resources available. Also, systems may have characteristics that vary with age, such as joint friction or tire pressure, or may even have minor sensor and actuator failures. Time varying reward functions are introduced whenever the system's goals are modified, as in response to an operator command. Despite the importance of the dynamic RL problem, no other published solutions exist. The nature of the dynamic RL problem—that the only information available is the robot's action-state history—suits it well to a case-based reasoning approach.

2 Method

S-learning operates by recording sequences of state-action pairs. The resulting libraries contain a reduced version of the system's history, a system memory. The memory can then be used to make predictions and guide the selection of the system's actions. When the system encounters a previously-experienced state, it retrieves sequences beginning with that state. The system can then re-execute the actions of recalled sequences that terminate in a reward state, as in a case-based approach.

2.1 S-learning algorithm

S-learning handles state-action (s-a) pairs, σ. An ordered sequence of state-action pairs is called a *sequence*, ϕ, and an unordered collection of ϕ is a *library*, κ. Both ϕ and κ may have any length of one or more, given by n_σ (number of state-action pairs) and n_ϕ (number of sequences), respectively.

An S-learning implementation can be broken into three main function blocks: the Agent, the Environment, and the Sequence Library. (Figure 1.)

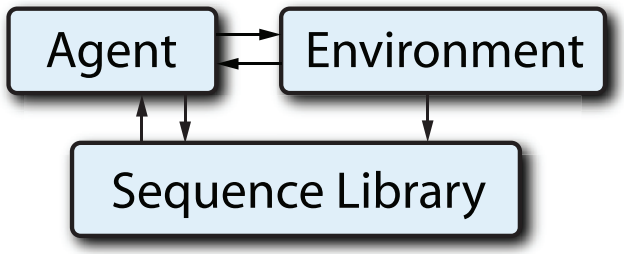

Fig. 1. Block diagram of S-learning. The Environment represents the system dynamics, f, and the Agent contains the reward function, g. The Sequence Library is created from the time history of s-a pairs.

Environment The Environment is the embodiment of the system dynamics, f (Equation 1). It receives action commands from the Agent and reports its state to Sequence Library and back to the Agent. In practice the Environment may be a continuous-time system, as long as it includes a means to execute discrete-time commands, a, and to report discrete-time sensor information, s.

The formulation of the dynamic RL problem places no constraints on the Environment. It may contain its own internal control system, stochastic elements, and learning capabilities. The Environment may do a large amount of

pre-processing on its sensor data and return highly-interpreted information. Alternatively, it may return nearly raw sensor data, binned and discretized in time. It may be physical or simulated, and there are no explicit limits to the complexity it can have.

Agent The Agent contains the reward function, g, and uses it to evaluate the plan candidates it receives from the Sequence Library. It executes the plans it selects by passing the corresponding actions to the Environment. The procedure the Agent follows during its operation is outlined below:

1. Define a target, τ, consisting of the most recent σ.
2. Query the Sequence Library for sequences that begin with τ, $\phi(\tau)$. The set of these form $\kappa(\tau)$, a collection of candidate plans.
3. Select a plan to execute from $\kappa(\tau)$:
   (a) Select the candidate plans that maximize the expected reward, $\overline{r}$, from the states that follow τ in each $\phi(\tau)$.
   (b) If there are more than one of these, select the shortest among them, that is, minimize n_σ.
   (c) If there is still more than one candidate, randomly select from among the remaining candidate plans such that a single plan, $\hat{\phi}$, is selected.
4. Execute the actions, a, associated with each element of $\hat{\phi}$.
5. Return to step 1.

The Agent also passes copies of the actions it executes, a, to the Sequence Library, so that it can assemble each a-s pair into a σ.

Sequence Library The Sequence Library is at the heart of S-learning. It allows S-learning to learn from its experience, use new learning as it is gained, generalize that learning to unfamiliar situations, make predictions, and attain goals. It has two primary functions: to pass candidate plans to the Agent and to record state space trajectories as they are observed. Candidate plans, $\phi(\tau)$ are selected on the basis of whether they begin with the target subsequence, τ, passed in by the Agent. The set of $\phi(\tau)$, $\kappa(\tau)$, is returned to the Agent. The process for recording newly observed states in the library is described below.

Due to the fact that S-learning is an experience-based learning algorithm, there is no distinction between memory and learning. Both are accomplished by the storage of sequences. As the Agent passes in actions, a, and the Environment passes in output states, s, the Sequence Library assembles them into a-s pairs, σ. A working memory of the most recently observed states is maintained. Sequences, ϕ, of length n_σ are stored in the library, κ. For ϕ_j that begins with σ_i, ϕ_{j+1} will begin with σ_{i+1}, that is, the subsequent sequences overlap by $n_\sigma - 1$ states. Through this accrual process, κ becomes the repository of the system's experience.

2.2 Robot Simulation

The S-learning algorithm was coded in Java and demonstrated with a simple simulated system. The simulation consisted of a mobile wheeled robot with two light sensors and eight contact sensors. (Figure 2) The robot occupied a 25×25 cell grid world. The robot could be positioned in the center of any one of the 625 cells in any one of the eight directions of the compass rose (up, down, right, left, and the directions offset 45 degrees from them). Contact sensors were located on the front, rear, sides, and corners of the robot. These registered whether the robot was in the border rows of the grid and in which direction were the contacted wall(s). The steering angle of the robot was also fed back with the sensory data.

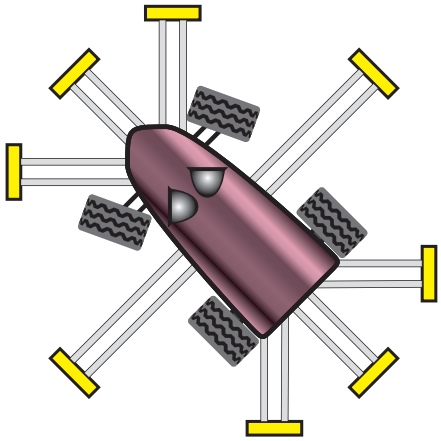

Fig. 2. Representation of the simulated robot. The steering angle of the front wheels, wall contact in any of 8 directions, and the sensed intensity at the two light sensors were all included in the state vector information.

A light source occupied another one of the grid cells and provided input to the light sensors. The two light sensors were each oriented 45 degrees from the robot's heading, one on the right and one on the left. The intensity of the light reaching the robot was the inverse of the square of the Euclidean distance from the robot to the light source and was determined by the following equation:

$$I = \frac{1}{(x_s - x_r)^2 + (y_s - y_r)^2 + \epsilon} \tag{4}$$

where x_r, y_r, x_s, and y_s are the $x-$ and $y-$coordinates of the robot and light source, respectively. $\epsilon = 10^{-7}$ was added to denominator to maintain numerical stability. The off-angle sensitivity, Ω, for each sensor was determined by the square of the cosine of the off-angle. This yielded a sensitivity of one in the direction of the sensor and a sensitivity of zero at an off-angle of $\pm$ 90 degrees.

Sensitivity at greater angles was also zero. The sensed intensity for each sensor, $\hat{I}$ was the product of the intensity and the off-angle sensitivity:

$$\hat{I} = I\Omega \tag{5}$$

The steering angle could have one of three values—straight ahead or 45 degrees to the right or left—and the robot could either drive forward or reverse. Steering and locomotion commands were incremental. A 'steer right' command increased the steering angle by 45 degrees unless it was already at its maximum value. 'Steer left' worked similarly. Movements forward and backward were made in one-cell increments. When the robot's heading was diagonal to the grid array, movements were made to the nearest diagonal cell. Motion was executed by first rotating the robot in place by the steering angle before moving either forward or backward. When it attempted to drive into a wall head-on or into a corner, the robot was not permitted to do so and it remained where it was. When it attempted to drive into a wall at a 45 degree angle, it instead moved to the next cell along the wall and maintained its heading. The robot was also not permitted to drive backward through the light source. Simultaneous 'steer right' and 'steer left' commands resulted in no change to the steering angle, and simultaneous 'forward' and 'reverse' commands resulted in no locomotion.

Action-state pair vector, $\boldsymbol{\sigma}$ The σ vector at each timestep was composed of binary elements representing the command issued and the sensory state after executing the command. The composition of σ is detailed in Table 1. The light sensor data was a binned $\hat{I}$, with the bin number given by the following:

$$b = 50 - \text{ceil}\left(\frac{1}{(\hat{I} + \epsilon)^{0.5}}\right) \tag{6}$$

The actual bin number, $\hat{b}$, was limited to the bins available:

$$\hat{b} = \begin{cases} 1 & \text{if } b < 1 \\ 50 & \text{if } b > 50 \\ b & \text{otherwise} \end{cases} \tag{7}$$

$\hat{b} = 1$ corresponded to very little to no light reaching the sensor, and $\hat{b} = 50$ corresponded to very intense light exposure. Both light sensors simultaneously achieved $\hat{b} = 50$ only when the robot drove forward into the light source. Vector elements corresponding to active sensor or command elements were equal to one. All others were zero.

Reward The goal of the system emerged from the nature of the reward. A reward vector, ρ, was created with the same length as σ, such that the total reward, r, was given by the following:

$$r = \prod_i \rho_i \text{ for all } i \text{ where } \sigma_i = 1 \tag{8}$$

Table 1. Composition of σ for the simulated robot.

Sensory modality or command type	Number of elements
Command: steering	2
movement	2
Sensors: contact	8
steering angle	3
light (right)	50
light (left)	50
Total:	115

In this formulation, ρ served as a set of multiplicative weights for σ. Sensory states were rewarded or penalized by assigning higher or lower values of ρ. The ρ used in the simulation was constructed in the following way:

- ρ values for steering angle positions were all set to 1, so as to neither reward nor penalize them.
- ρ values for contact sensors were set to 0.7 to penalize contact with the borders of the grid.
- For the light sensors, the 50 reward vector elements that corresponded to the gradations in intensity were set according to the relation $\rho_i = i/50$. The most intensely sensed light produced a ρ of 1, resulting in no penalty, while the weakest sensed light produced a ρ of 0.02, a strong penalty.

Sequence library creation At each timestep, the action that was executed and the state that resulted from that action were combined into a state-action pair, σ. The sequence of $n_\sigma^{\max}$ most recently observed sequences was maintained, where $n_\sigma^{\max}$ was the maximum sequence length, a parameter manually set in software. As described above, the longest sequence not in the library already (up to the maximum sequence length) was added to the Sequence Library. Due to the simplicity of the system, all the information necessary to make reasonably accurate predictions about the system was available at each timestep. In this case a maximum sequence length of $n_\sigma^{\max} = 2$ was sufficient. More complex systems would benefit from a greater $n_\sigma^{\max}$, as it would be able to compensate somewhat for partial or noisy state information.

As sequences were added to the library, they were assigned an initial strength value, 10^6. At each timestep, the strength was decreased by 1. The strength of each sequence was multiplied by 10 after each repeat observation. If strength ever decayed to 0, the sequence was dropped from the library. This provided a mechanism for rarely observed sequences to be forgotten. The deterministic nature of the simulated system did not need to make use of this (hence the large initial strength), but it is a feature of S-learning that suits it for use with more complex systems as well. It can also be seen that after several repeated

observations a sequence's existence in the library would be assured for the life of the system. This is analogous to recording an experience in long-term memory.

Action selection The Agent referred to the Sequence Library to help determine which action command to send at each timestep. All sequences that began with the most recent state were used as a set of predictions. (The most recent state might be contained in multiple σ's, since several actions may have resulted in that state in the system's history. Sequences beginning with all σ's matching the most recent state were returned.) Each sequence represented a possible future. The Agent compared the reward at the final state of each sequence to the reward at the initial state, and the sequences with the greatest increase in reward were selected as the most promising.

The actions pertaining to each sequence defined a plan. By executing the actions in the same order, it was possible to create the same sequence of states. However it was not guaranteed to do so. Some state information, such as distance to the grid borders when not in contact with them, was not directly sensed and so introduced some variability into the effects produced by a given series of actions. Although it was a relatively minor effect with the simulated robot, with more complex systems containing more limited state information, the variability of the effects of a given action would increase greatly. The most promising sequences found in the Sequence Library represented the best case scenarios for each plan. In order to make a more informed decision, the expected value of the final reward for each plan (up to 50 of them) was calculated in the following way.

The library was queried for all the sequences starting from the most recent state and executing each plan. The final rewards for the sequences executing a given plan were averaged, weighted by the log of the strength of each sequence:

$$\overline{r} = \frac{\sum_i r_i \log(\omega_i + 1)}{\sum_i \log(\omega_i + 1)} \tag{9}$$

where $\overline{r}$ is the weighted average reward and r_i is the reward and ω_i is the strength associated with each sequence. One was added to ω_i to ensure that the log remained non-negative.

$\overline{r}$ represented the expected value of the reward a given plan would produce. The plan with the highest value of $\overline{r}$ was selected for execution, given that $\overline{r}$ was greater than the reward at the most recent state. In the case of the simulated robot, with a maximum sequence length of two, the plan always consisted of a single action; that action command was passed to the robot at that timestep.

If no plans were expected to increase the reward, then the robot generated a random exploratory plan. Exploratory plans were also initiated at random intervals (on average one out of sixty timesteps). In addition, a 'boredom' condition was met if the same state was observed more than five times within the last fifty timesteps. This also resulted in the creation of an exploratory plan. Exploratory plans were of random length, up to 4 actions long. Each action was randomly generated with each of the 4 elements of the action vector having an independent, 25% chance of being active. Exploration provided some random variation

to S-learning's operation, allowing it to explore its environment and avoid getting caught in highly non-optimal behavior patterns, such as infinitely-repeating cycles.

Task structure The robot was able to increase its reward by orienting itself toward the light source and approaching it. When the robot drove forward into the light source, both light sensors registered their maximum intensity, and generated the maximum reward. This was defined to be the completion of the task. After the task was completed, the robot and the light source were both randomly repositioned in the grid and the task began again. The measure of the performance in each run was the number of timesteps required to complete the task.

3 Results

On the first run the robot did not approach the light source in any direct way. The beginning of the run is shown in Figure 3. Initially, the Sequence Library was empty and all movements were random and exploratory. Learning gained during the first movements was used as soon as it was applicable. Notably, the somewhat direct descent from the upper-right region of the grid to the lower-middle region was the repeated application of single successful movement.

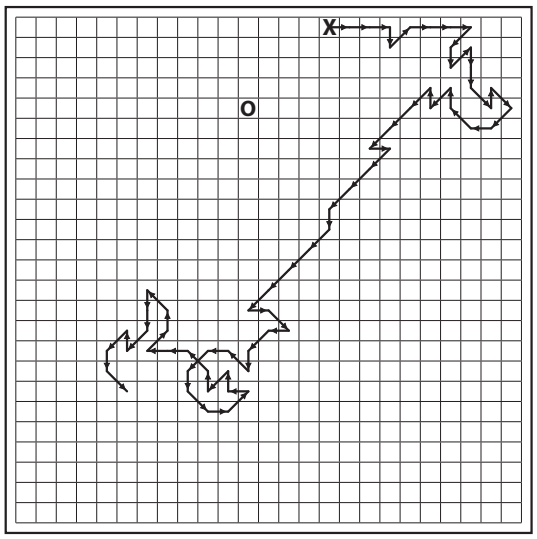

Fig. 3. The initial movements of a typical naïve simulation run.

The earliest runs consisted mostly of exploration and were relatively lengthy. The first twenty runs averaged over 350 timesteps per run. As the Sequence

Library became more complete and the state space was better explored the number of timesteps required to reach the goal decreased rapidly. (Figure 4) At 200 runs, the average number of timesteps had decreased to 50. From that point, performance continued to improve, but much more slowly. After 1400 runs, a typical run lasted 40 timesteps.

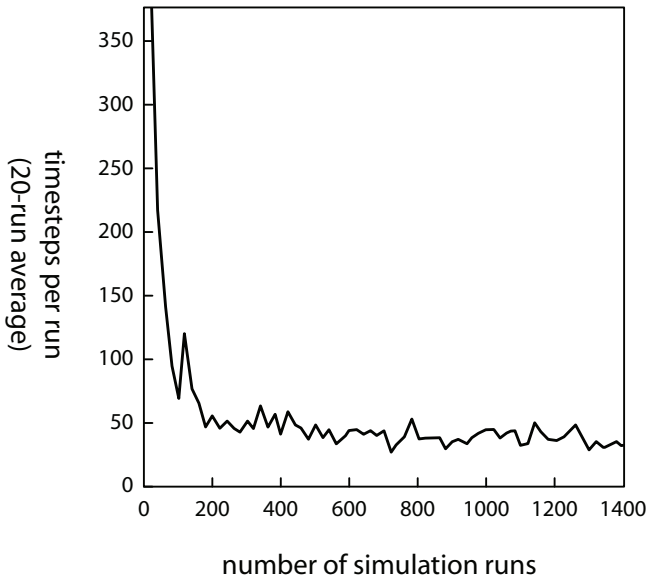

Fig. 4. Goal acquisition performance. The average performance is shown for 20-run blocks. Running on a 2.66 GHz Intel Xeon Processor with 8 GB of RAM under a 64-bit Linux, these data were generated in 16 minutes.

Later runs show a much more direct approach to the goal. Figure 5a shows an optimally direct run. Figure 5b and c show nearly direct runs that have been interrupted by exploratory movements. Occasionally the robot wandered for a time before closing in on the goal, particularly when it began far from the goal, with its sensors facing away from it. This, together with exploratory interludes, caused the average performance to stay as high as it did, rather than drop to optimal levels—closer to 20.

4 Discussion

This work has demonstrated the implementation and operation of S-learning, a model-free learning and control approach that is fundamentally case-based. S-learning was able to learn to control a simple robot in a simple environment.

S-learning is capable of addressing some dynamic reinforcement learning problems, although it should be noted that the light-seeking robot simulated

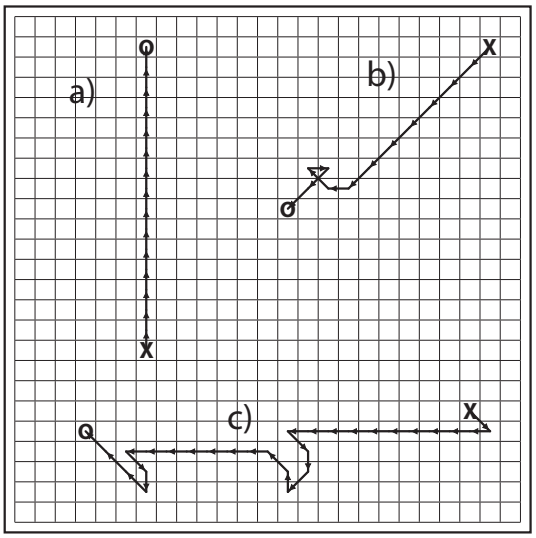

Fig. 5. Three typical simulation runs after 1400 runs.

here is not one. The addition of multiple light sources of different colors, and time varying rewards associated with different colors would be an example of a dynamic RL problem. For examples of S-learning solving dynamic RL problems, see [2, 8].

The two degree-of-freedom, non-holonomic mobile robot simulated here could be modeled with a trivial amount of effort. In fact, this model existed in the simulation in order to generate the appropriate behavior. However, S-learning didn't make use of that model (except for the structured sensory information that the simulation produced), but treated the robot system as a black box. The key aspect of S-learning's operation is that it relied only on the system's prior experience, rather than on *a priori* knowledge. This simulation does not showcase the extent of S-learning's capability. Rather, it's purpose was to detail an instantiation of the S-Learning algorithm.

4.1 Limitations of S-learning

The robustness and model-independence of S-learning comes at a price. The largest cost is in long learning times. Significant training time, approximately 75,000 timesteps, was required to learn to control a relatively simple system. This raises the question of when it would be appropriate to use S-learning. In any implementation where a model is available, the trade-off between which portions to learn and control with S-learning and which to control with a more conventional model-based controller is a trade-off between learning time (short-term performance) and robustness (long-term performance). This question can only be answered based on the specific goals and constraints of each implementation.

Some of the details of S-learning's implementation are specific to the system. One of these details is the maximum sequence length, $n_\sigma^{\max}$. As described previously, $n_\sigma^{\max} = 2$ was known to be appropriate to the simulation due to its determinism and simplicity. However, other systems may benefit from larger values of $n_\sigma^{\max}$. Humans' capability to remember 7 ± 2 chunks of information suggest that $n_\sigma^{\max} = 7$ is an estimate with reasonable biological motivation. Similarly the dynamics of sequence strength, underlying consolidation and forgetting of sequences, may need to be varied to achieve good performance on different systems. Initial tests show that the most critical design decisions in an S-learning implementation are the discretization of sensor data and the assignment of reward vectors that produce desirable behaviors. Some primary considerations when discretizing sensors are discussed in [8, 9], but additional work is required to fully identify the trade-offs involved.

4.2 Implications

Due to its model agnosticism, S-learning's case-based reasoning approach to robot control is potentially applicable to hard problems, such as bipedal locomotion and manipulation. In the case of locomotion, the system model can be extremely, if not intractably, complex, and environments may be completely novel. In addition, extra-laboratory environments can be harsh, and insensitivity to sensor and actuator calibration may be desirable as well. In the case of manipulation, mathematical modeling of physical contact is notoriously difficult and requires a lot of computation to perform well. It also requires high-fidelity physical modeling of the entire system, which is not possible when handling unfamiliar objects. These attributes suggest that locomotion and manipulation are two examples of hard problems to which S-learning may provide solutions.

Acknowledgements

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

1. J. Albus. A new approach to manipulator control: Cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97:220–227, 1975.
2. S. Hulet, B. Rohrer, and S. Warnick. A study in pattern assimilation for adaptation and control. In *8th Joint Conference on Information Systems*, 2005.
3. L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabalistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
4. S. E. Levinson. *Mathematical Models for Speech Technology.* John Wiley and Sons, Chichester, England, 2005. pp. 238-239.

5. J. L. McKinstry, G. M. Edelman, and J. L. Krichmar. A cerebellar model for predicitive motor control tested in a brain-based device. *Proceedings of the National Academy of Sciences*, 103(9):3387–3392, 2006.
6. A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.
7. J. Morimoto, G. Zeglin, and C. G. Atkeson. Minimax differential dynamic programming: Application to a biped walking robot. In *Proceedings of the IEEE/RSJ Intl. Conference on Intellignet Robots and Systems*, pages 1927–1932, 2003.
8. B. Rohrer. S-learning: A biomimetic algorithm for learning, memory, and control in robots. In *Proceedings of the 3rd International IEEE EMBS Conference on Neural Engineering*, 2007.
9. B. Rohrer. Robust performance of autonomous robots in unstructured environments. In *Proceedings of the American Nuclear Society 2nd International Joint Topical Meeting on Emergency Preparedness and Response and Robotics and Remote Systems*, 2008.
10. R. S. Sutton. *Planning by incremental dynamic programming*, chapter Proceedings of the Eighth International Workshop on Machine Learning, pages 353–357. Morgan Kaufmann, 1991.
11. D. C. Tarraf, A. Megretski, and M. A. Dahleh. A framework for robust stability of systems over finite alphabets. *IEEE Transactions on Automatic Control*, June 2008. To appear as a regular paper in the IEEE Transactions on Automatic Control (scheduled for June 2008).
12. Y. Tassa, T. Erez, and B. Smart. *Advances in Neural Information Processing Systems*, chapter Receding horizon differential dynamic programming, pages 1465–1472. MIT Press, Cambridge, MA, 2008.
13. C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.